Universidade Federal de Pernambuco
Graduação em Ciência da Computação
Centro de Informática

Proposta de Trabalho de Graduação

Integrando UML e Métodos Formais

Aluno: Rafael Magalhães Borges (rmb2@cin.ufpe.br)
Orientador: Alexandre Cabral Mota (acm@cin.ufpe.br)
Co-orientador: Augusto Cézar Alves Sampaio (acas@cin.ufpe.br)

Recife, 1 de Junho de 2004.

1 Contexto

A crise de software é, ainda hoje, o maior problema da Engenharia de Software. Diversos projetos são cancelados e outros tantos estouram custos e prazos. Poucos são aqueles concluídos com sucesso, sem erros ou atrasos. Este problema se agrava quando falamos dos sistemas considerados críticos: sistemas que envolvem elevadas somas de dinheiro ou vidas humanas. Um erro pode ser fatal, como o caso do Therac-25, que aplicava doses radioativas letais nos pacientes [18], ou caríssimo, como o do Ariane 5, cujas falhas de lançamento custaram milhões ao programa espacial europeu [15].

Alguns trabalhos [11, 4, 3] sugerem que as causas desta crise são a instabilidade dos requisitos e a complexidade inerente do software. Outros [13], mais filosóficos, comparam os desenvolvedores atuais aos artesãos pré-industriais: ambos produzem seus artefatos utilizando técnicas baseadas no empirismo, desconhecendo a ciência por trás dos seus ofícios. Inclusive, [7] vai além: sugere não só o embasamento, mas também a verificação formal dos programas. Vale ressaltar que todos concordam que produzir software de qualidade (cuja maior componente é a corretude) é essencial.

Diversas *balas de prata* para essa crise foram propostas: ferramentas especializadas, orientação a objetos e inteligência artificial são exemplos disso [3]. Dentre elas, Métodos Formais merecem destaque. Estabelecer a matemática como alicerce do desenvolvimento de software dá rigor científico a diversos conceitos outrora informais; é agora possível verificar formalmente a corretude dos programas através de provas.

Noções de refinamento passaram a ter uma importância muito grande: especificar um sistema utilizando estruturas matemáticas e abstraindo detalhes operacionais permite um maior entendimento do problema. A prova das propriedades dos sistemas passa a ser bastante intuitiva e relativamente fácil. Depois, a substituição dessa especificação por outras mais concretas (cujas relações com a original sejam demonstradas), com estruturas de dados cada vez mais computacionais e menos abstratas, garante as propriedades do modelo original [32]. Por fim, a aplicação de transformações (ou leis) matemáticas leva-nos à derivação de uma implementação que, por construção, está correta [20, 8].

Contudo, mesmo com casos de sucesso como o CICS [32], TCAS II [12] e muitos outros [1], Métodos Formais ainda não são utilizados em larga escala na indústria. Como destacam [6, 30], isso se deve à sua forte notação matemática e ao uso de ferramentas ineficientes e não-amigáveis. Porém, é importante frisar que nenhum método foi universalmente aceito até hoje.

Para suprir essas deficiências e tornar Métodos Formais mais acessíveis, existem duas alternativas mais comuns: a simplificação da linguagem formal,

o que inclui até a adição elementos gráficos [14], ou a utilização de uma outra linguagem, geralmente informal, mapeando-a para uma formal [21, 26, 33].

Apesar dessa primeira alternativa possuir seus méritos, acreditamos que a segunda é mais promissora. Podemos assim utilizar, como intermediárias, as linguagens com as quais o desenvolvedor já está habituado e mapeá-las numa linguagem formal mais poderosa, concebida sem maiores restrições conceituais (embora sejam necessárias algumas para a utilização prática).

Como linguagem intermediária, UML [22, 24] aparece como a opção mais importante. Ela utiliza elementos gráficos para representar as diversas entidades assim como seus relacionamentos e graças a essa simplicidade, aliada à facilidade de entendimento, tornou-se o padrão do mercado. Porém, é uma linguagem informal (portanto ambígua) e insuficiente para representar até propriedades mais simples [23].

Já para o papel de linguagem formal, OhCircus [5] é uma excelente escolha. OhCircus integra conceitos bem estabelecidos na comunidade formal: a linguagem baseada em modelos Z [31], a álgebra de processos CSP [27] e o cálculo de refinamentos [20], além dos conceitos de orientação a objetos, provendo uma linguagem unificada para classes e processos. Vale destacar que algumas idéias foram obtidas a partir de UML-RT[1] [19, 29], o que a torna ainda mais apropriada para esse mapeamento.

Atualmente, essa integração entre linguagens de mercado e formais é uma área de pesquisa bastante ativa, permitindo que as práticas da Engenharia de Software sejam provadas formalmente, enquanto os resultados dos Métodos Formais são aplicados na indústria. Em particular, *hidden formal methods* vai além, utilizando na prática os resultados estabelecidos formalmente, mas sem que o desenvolvedor perceba o uso de Métodos Formais.

2 Objetivos

O objetivo deste trabalho é capturar os principais elementos que constituem os diagramas de classes anotados de UML e mapeá-los em especificações OhCircus. Este é um tópico de pesquisa bastante ativo [2, 16, 17, 25], mas nossa abordagem difere das demais porque utiliza uma linguagem projetada para acomodar várias construções de UML(-RT) e procura preservar a estrutura do diagrama de classes. Vale destacar que a semântica de OhCircus ainda está incompleta; porém, como as vantagens de utilizá-la superam as desvantagens, esperamos que esse trabalho seja uma contribuição para a evolução da própria linguagem.

[1] UML-RT é uma extensão de UML que contempla concorrência.

Como discutido no contexto, UML é por si só insuficiente para capturar todas as propriedades relevantes do sistema [23], sendo também necessárias anotações (invariantes, pré- e pós-condições *etc*). Por outro lado, a linguagem proposta pela OMG para suprir este papel, OCL [23], é limitada: apenas especifica restrições (semelhantes a asserções) e ainda não possui semântica formal bem definida. Neste trabalho, consideraremos algumas características e construções dessa linguagem, mas OhCircus será utilizada nas anotações.

Além de tratar dos elementos individuais do diagrama de classes (as próprias classes), também preservamos toda a sua estrutura, como relacionamentos, propriedades globais e aspectos dinâmicos do sistema (via *system history* [10]). Isto é realizado através de uma (meta-)classe, sintaticamente equivalente às demais, que concentra tal estrutura (*classe-modelo*). A principal motivação para isso é explorar refinamento em UML [28].

Por fim, diversos trabalhos [2, 9, 17], inclusive [24], tomam como verdade a noção de equivalência entre associações e atributos. Utilizando nossa abordagem, propomos demonstrar que as associações representam uma visão abstrata do diagrama de classes, enquanto a interpretação delas como atributos é um dos refinamentos possíveis. O objetivo principal é, além do próprio resultado, dar uma intuição de (e possivelmente consolidar) o mapeamento e a noção de classe-modelo.

É importante ressaltar que este trabalho é apenas um passo inicial em direção ao mapeamento completo de UML-RT para OhCircus. Várias pesquisas podem ser exploradas a partir desta, como a construção do mapeamento inverso (a partir da preservação das características) e futuras extensões, contemplando, por exemplo, os aspectos dinâmicos (o uso de uma classe-modelo) e OCL.

3 Cronograma

	Mês											
Atividade	*Junho*				*Julho*				*Agosto*			
Investigar classes-modelo	*	*	*	*	*							
Preparar o mapeamento					*	*	*	*				
Abordar o refinamento								*	*	*	*	*
Elaborar o relatório			*	*	*	*	*	*	*	*	*	*
Elaborar a apresentação											*	*

Tabela 1: Cronograma

Referências

[1] J. Bowen and M. Hinchey. *Applications of Formal Methods*. Prentice Hall PTR, 1995.

[2] R. Breu, U. Hinkel, C. Hofmann, C. Klein, B. Paech, B. Rumpe, and V. Thurner. Towards a Formalization of the Unified Modeling Language. In *Proceedings of ECOOP'97*. Springer Verlag, LNCS, 1997.

[3] F. Brooks, Jr. No Silver Bullet: Essence and Accidents of Software Engineering. *Computer Magazine*, 20(4):10–19, 1987.

[4] F. Brooks, Jr. *The Mythical Man-Month (Anniversary ed.)*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[5] A. Cavalcanti, A. Sampaio, and J. Woodcock. A Unified Language of Classes and Processes. In *St Eve: State-Oriented vs. Event-Oriented Thinking in Requirements Analysis, Formal Specification and Software Engineering*, Satellite Workshop at FM'03, unknown 2003.

[6] E. Clarke and J. Wing. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys*, 1996.

[7] E. Dijkstra. The Humble Programmer. *Communications of the ACM*, 15(10):859–866, 1972.

[8] E. Dijkstra. *A Discipline of Programming*. Prentice Hall PTR, 1997.

[9] A. Evans. Reasoning with UML Class Diagrams. In *2nd IEEE Workshop on Industrial Strength Formal Specification Techniques*. IEEE, 1998.

[10] R. Gheyi and P. Borba. Refactoring Alloy Specifications. In *WMF 2003: 6th Workshop on Formal Methods, Brazil*, pages 166–181, 2003.

[11] W. Gibbs. Software's Chronic Crisis. *Scientific American (International ed.)*, 271(3):86–95, 1994.

[12] M. Heimdahl. Experiences and lessons from the analysis of tcas ii. *SIGSOFT Softw. Eng. Notes*, 21(3):79–83, 1996.

[13] C. Hoare. Programming: Sorcery or Science? *IEEE Software*, 1(2):5–12, 15–16, April 1984.

[14] D. Jackson. Micromodels of Software: Lightweight Modelling and Analysis with Alloy. Technical report, Software Design Group, MIT Lab for Computer Science, 2002.

[15] J.-M. Jézéquel and B. Meyer. Design by Contract: The Lessons of Ariane. *IEEE Computer*, 30(2):129–130, January 1997.

[16] S. Kim and D. Carrington. A Formal Mapping between UMLModels and Object-Z Specifications. *Lecture Notes in Computer Science*, 1878:2–21, 2000.

[17] K. Lano and J. Bicarregui. UML Refinement and Abstraction Transformations. *ROOM 2 Workshop, Bradford University*, 1998.

[18] N. Leveson and C. Turner. An Investigation of the Therac-25 Accidents. *IEEE Computer*, 26(7):18–41, June 1993.

[19] A. Lyons. UML for Real-Time Overview. Whitepaper, ObjecTime Limited, April 1998.

[20] C. Morgan. *Programming from Specifications (2nd ed.)*. Prentice Hall International (UK) Ltd., 1994.

[21] P. Moura, R. Borges, and A. Mota. Experimenting Formal Methods through UML. Submitted to WMF'2003, July 2003.

[22] OMG. UML 2 Infrastructure Final Adopted Specifcation. Whitepaper, Object Management Group, September 2003.

[23] OMG. UML 2 OCL Final Adopted Specification. Whitepaper, Object Management Group, October 2003.

[24] OMG. UML 2 Superstructure Final Adopted specification. Whitepaper, Object Management Group, August 2003.

[25] R. Paige. Integrating a program design calculus and a subset of UML. *The Computer Journal*, 42(2), 1999.

[26] D. Roe, K. Broda, and A. Russo. Mapping UML Models incorporating OCL Constraints into Object-Z. Technical Report 2003/9, Imperial College London, 2003.

[27] A. Roscoe, C. Hoare, and R. Bird. *The Theory and Practice of Concurrency*. Prentice Hall PTR, 1997.

[28] A. Sampaio, A. Mota, and R. Ramos. Class and Capsule Refinement for UML-RT. In *WMF 2003: 6th Workshop on Formal Methods, Brazil*, pages 16–34, 2003. Extended version to appear in Electronic Notes in Theoretical Computer Science, Elsevier, 2004.

[29] B. Selic and J. Rumbaugh. Using UML for Modeling Complex Real-Time Systems. Whitepaper, Rational Software Corp., March 1998.

[30] I. Sommerville. *Software Engineering*. Addison-Wesley, 2002.

[31] M. Spivey. *The Z Notation*. Prentice-Hall, 1992.

[32] J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice Hall, 1996.

[33] P. Zeppo. From UML to B Specifications. Master's thesis, Dept. of Computer Science, King's College London, 2002.

Datas e Assinaturas

1 de Junho de 2004

Alexandre Cabral Mota
(Orientador)

Augusto Cézar Alves Sampaio
(Co-orientador)

Rafael Magalhães Borges
(Proponente)